Embedded Artificial Intelligence

SETI Master

Lab 4 — Knowledge Distillation for Frugal Embedded AI

From numerical principles and controlled training to a measured KV260 implementation

Author:	Alaa Eddine Mazouz
Academic year:	2026–2027
Version:	5.14
Date:	26 October 2026
Recommended guided path:	MNIST, teacher trained in the lab
Recommended student:	LeNet student, width multiplier 1.0
Target:	AMD Kria KV260, DPU B4096

Télécom Paris

Work in the lab folder `TP04_distillation/`, which contains `scripts/` and `src/`.

Host shell. Type `bash` once in every new host terminal before the course commands that use `source`; the teaching hosts default to zsh. Commands launched with `bash script.sh` already select Bash.

Raw datasets. Raw data are shared in the course-root `datasets/` folder; a standalone lab uses `$HOME/.cache/c5/datasets` (or `$XDG_CACHE_HOME/c5/datasets`). Set `C5_DATA=/absolute/path` before sourcing `scripts/00_source_env.sh` to override it. Existing datasets stay in place; experiment outputs stay in this lab.

Purpose of this lab

Knowledge distillation (KD) trains a compact *student* with a stronger *teacher*: its class probabilities (response KD), its feature maps (here attention transfer), or both; the deployed model is the student alone. You observe the KD quantities on small tensors, train a teacher and five students with different objectives, compare them under controlled conditions, run them on the KV260, and finally write the KD code yourself. The theory (soft targets, temperature, the T^2 factor, loss weights, attention maps, fair comparisons) is in the L04 slides; this handout gives the commands and what to record.

Plan

Part	Content
1	concept pre-lab: KD quantities on small tensors (C1–C4)
2	teacher: data, capacities, training
3	five students, one network
4	analysis and reference check
5	KV260: quantize, compile, run, measure
6	your code: the KD primitives (C5), then the final task

Parts 1 to 5 only run provided scripts: the code you write is in Part 6, at the end, so it never blocks the pipeline. Start a training, then answer the checkpoint of the previous step while it runs.

Before starting

Each listing starts with a comment that says where it runs:

- `# host`: a terminal on the lab machine, in the `TP04_distillation` folder;
- `# container`: the Vitis AI shell opened by `02_enter_container.sh`, with PyTorch, the quantizer and the compiler (the prompt starts with `(vitis-ai-pytorch)`);
- `# KV260`: a shell on the board, opened with `ssh root@${BOARD#*@}` (Lab 2).

```
# host
cd TP04_distillation
TAG=your_mnist_kd bash scripts/00_setup_experiment.sh
TAG=your_mnist_kd bash scripts/02_enter_container.sh
```

```
# container: once per terminal, if you will use the KV260 (account root, see Lab 2)
export BOARD=maquette-07          # the name of your board, or its IP address
```

The container image is found as in the previous labs (`VAI_SIF`, then the shared lab copy, then `/tmp/$USER/vitis-ai-pytorch-cpu.sif`); if it is missing, the script prints the `apptainer pull` command to run once. For a second container shell, run `02_enter_container.sh` again in another terminal; in a container shell opened another way, activate the environment first:

```
# container
source /opt/vitis_ai/conda/etc/profile.d/conda.sh
conda activate vitis-ai-pytorch
```

The TAG and the rules of the experiment

- **One TAG = one experiment.** `00_setup_experiment.sh` fixes the dataset, the architectures and widths, the data subsets, the seeds and the default training and KD settings (guided setup: MNIST, LeNet teacher and student, 1,000 labelled images, 10,000 transfer images, $T = 2$, $0.3\,L_{CE} + 0.7\,L_{KD}$, attention weight 0.1). Every later command needs only `TAG=<name>`.
- **Overrides.** A training setting can be given on the command that trains a model, for example `TAG=<name> STUDENT_PATIENCE=10 bash scripts/04_train_student_ce.sh` (`STUDENT_LR`, `STUDENT_BATCH`, `STUDENT_PATIENCE`, `T`, `CE_WEIGHT`, `KD_WEIGHT`, `AT_WEIGHT`, ...; `TEACHER_` for the teacher). The values used are saved with the model; give the same values to all the models you compare.
- **Rerun.** A script never erases a trained model: it prints the command to replace it (`OVERWRITE=1`; the old result moves to `archive/`). A structural change (dataset, width, data budget, seed) is a new TAG.
- **Stopping.** No epoch limit by default: after 8 epochs, training stops when validation accuracy has not improved for 6 epochs (`STUDENT_PATIENCE`), the best validation checkpoint is restored and the test set is evaluated once.
- **Results.** `TAG=your_mnist_kd bash scripts/00_show_results.sh` reprints every table (`MODE=model MODEL=student_ce` for one model).

1 Concept pre-lab — observe KD before training a network

Small tensors, every intermediate value printed (formulas in L04): predict each number before reading it. The scripts need a TAG name only and write to `out/runs/<TAG>/concepts/`; a variant gets its own TAG.

1.1 C1 — hard labels, soft labels, and temperature

```
# container
TAG=your_mnist_kd bash scripts/02a_concept_soft_labels.sh
TAG=your_concept_logits2 LOGITS="6,4,1,-1" bash scripts/02a_concept_soft_labels.sh
```

For each temperature: every class probability, the argmax, the maximum probability, the non-target mass and the normalized entropy.

Checkpoint C1 — reason from the softmax

Before reading the output, predict how the maximum probability and the entropy change from $T = 1$ to $T = 4$. Then:

1. Why does dividing every logit by the same positive T preserve the argmax? What happens to the ratios p_i/p_j when T increases?
2. What remains when $T \to \infty$? Why can a very large temperature make KD less useful?

1.2 C2 — what CE sees and what response KD adds

```
# container
TAG=your_mnist_kd bash scripts/02b_concept_response_kd.sh
```

Two students with the same probability on the correct class (same CE) but different probabilities on the wrong classes (different KD); the CE, KD and combined gradients at one student; the KD loss and gradient norm with and without T^2 for several temperatures.

Checkpoint C2 — distinguish CE and response KD

1. How can two students have exactly the same CE but different KD losses? Why is training on `argmax(teacher)` no longer response KD?
2. Compare the printed gradient norms with and without T^2. Which practical problem does the factor avoid when you change the temperature?

1.3 C3 — feature matching and attention transfer

```
# container
TAG=your_mnist_kd bash scripts/02c_concept_attention.sh
```

Attention maps (L04) of an aligned, a spatially shifted and a globally rescaled student.

Checkpoint C3 — what does attention preserve?

1. Which information is removed when the channels are collapsed into one map? Which dimensions must still match between teacher and student?
2. Explain with the printed numbers why a global rescaling of the features has little or no effect.
3. A student reduces its attention distance but its test accuracy does not improve. Give two possible reasons.

1.4 C4 — loss weights and optimization priorities

The objective is $L = w_{CE}L_{CE} + w_{KD}L_{KD} + \beta L_{AT}$.

```
# container
TAG=your_mnist_kd bash scripts/02d_concept_loss_balance.sh
TAG=your_concept_weights2 CONCEPT_CE_WEIGHT=0.5 CONCEPT_KD_WEIGHT=1.5 \
  CONCEPT_AT_WEIGHT=0.2 bash scripts/02d_concept_loss_balance.sh   # if time permits
```

For four weight settings: the raw losses, the weighted contributions, their shares, and a few small gradient steps.

Checkpoint C4 — weights are not percentages

1. If $w_{KD} > w_{CE}$, must the KD contribution be larger than the CE contribution? Answer with the printed raw losses.
2. What would you expect if CE is almost removed? What risk appears if the teacher is wrong or poorly calibrated?

2 Teacher and student

The guided experiment uses MNIST, small enough to train the teacher during the lab. Check the setup and prepare the data: MNIST is downloaded once, into the dataset folder of the course (`data/` links to `datasets/` next to the lab folders, or to the course's shared copy).

```
# container
TAG=your_mnist_kd bash scripts/01_check_setup.sh
TAG=your_mnist_kd DOWNLOAD=1 bash scripts/01_prepare_data.sh
```

The setup table gives the input shape, the dataset sizes, and the capacities, parameters and MACs of the teacher and the student:

Model	Convolution channels	Hidden width	Use
Teacher, width 1.0	$16 \to 32$	128	supervision during training
Student, width 1.0	$4 \to 8$	32	guided student
Student, width 0.5	$2 \to 4$	16	small-student study (new TAG)
Student, width 0.25	$1 \to 2$	8	extreme-capacity study (new TAG)

Train the teacher from scratch:

```
# container
TAG=your_mnist_kd bash scripts/03_train_teacher.sh
```

The teacher is selected on validation accuracy (5,000 held-out training images). Before any student exists in the TAG it can be retrained with other settings; after that, it is locked:

```
# container: optional, only to retrain the teacher
TAG=your_mnist_kd OVERWRITE=1 TEACHER_PATIENCE=10 TEACHER_LR=0.0005 \
  bash scripts/03_train_teacher.sh
```

Teachers for Fashion-MNIST and CIFAR-10 are also supplied (Appendix A).

Checkpoint — is the teacher appropriate for KD?

Record the teacher's validation and test accuracy, parameters and MACs. Why should the teacher normally be stronger than the student? What could happen if it is systematically wrong, or only marginally better? Why choose training settings on validation rather than test accuracy?

3 Five students, one network

3.1 What changes and what does not

All five students are the same network, trained from the same initial weights (the trainer resets the seed before building the student and records a hash of the initial weights), with the same validation split, optimizer and stopping rule. The guided CE run uses the labelled subset; KD also uses transfer examples. This compares objective and data availability together. The final task controls both methods to the same examples.

The guided experiment is **data-efficient**: only 1,000 training images have a label, and the teacher's outputs are available on 10,000 transfer images (which include the 1,000 labelled ones). CE uses the labelled images only; the teacher terms use all transfer images. With 1,000 labels, the CE student reaches only about 88–89 % on MNIST: this is expected, and it is the gap that KD can fill.

Student	Objective (default weights)	Labels	Teacher outputs	Teacher features	Script
CE	L_{CE}	1,000	–	–	`04`
KD-only	L_{KD}	–	10,000	–	`06a`
CE+KD	$0.3\,L_{CE} + 0.7\,L_{KD}$	1,000	10,000	–	`06b`
CE+AT	$0.3\,L_{CE} + 0.1\,L_{AT}$	1,000	–	10,000	`06d`
CE+KD+AT	$0.3\,L_{CE} + 0.7\,L_{KD} + 0.1\,L_{AT}$	1,000	10,000	10,000	`06c`

Model IDs, used by `MODEL=` and as result folder names: `teacher`, `student_ce`, `student_kd_only`, `student_ce_kd`, `student_attention` (CE+AT), `student_kd_attention` (CE+KD+AT).

Each epoch prints the train and validation accuracy and the weighted CE, KD and AT terms (`*`: new best); at the end, the model report and the comparison of all models trained so far. The selected epoch is usually not the last one.

3.2 Supervised baseline (CE)

```
# container
TAG=your_mnist_kd bash scripts/04_train_student_ce.sh
```

Checkpoint — baseline evidence

Record the selected epoch, epochs run, best validation accuracy, test accuracy, updates, samples processed, training time, parameters, MACs and initial-weight hash. Why must a CE/KD comparison use the same student graph and initialization? Why may the selected epoch differ between methods under a common validation-based stopping rule, and what extra control would a fixed-compute comparison need?

3.3 Response KD (CE+KD and KD-only)

The response term is the one of L04 (`src/kd_losses.py`). The teacher is frozen and the images are not augmented, so its outputs are computed once and cached:

```
# container
TAG=your_mnist_kd bash scripts/05_cache_teacher_targets.sh
TAG=your_mnist_kd bash scripts/06b_train_student_ce_kd.sh
TAG=your_mnist_kd bash scripts/06a_train_student_kd_only.sh
```

In the epoch history, `RAW KD` is L_{KD} and `WT KD` is $w_{KD}L_{KD}$. KD-only removes the ground-truth anchor: it shows what the teacher responses alone carry.

Checkpoint — response KD

At early, middle and late epochs, compare the raw and weighted CE and KD terms. Does $w_{KD} = 0.7$ mean that KD gives 70 % of the optimization signal? Compare CE, KD-only and CE+KD in terms of the information each receives. If CE+KD reaches its best validation accuracy earlier than CE, what does that show, and what does it not yet prove about the converged result?

3.4 Attention transfer (CE+KD+AT and CE+AT)

`forward_with_features` returns the logits and the feature maps of the selected stages (same resolution in teacher and student here). The attention students run the frozen teacher at every step, so they train more slowly; the deployed graph is still the student alone.

```
# container
TAG=your_mnist_kd bash scripts/06c_train_student_ce_kd_at.sh
TAG=your_mnist_kd bash scripts/06d_train_student_ce_at.sh
```

CE+AT is the feature-only ablation: CE supplies the labels and attention is the only teacher signal (its `RAW KD` and `WT KD` columns show `--`). There is no AT-only student: spatial alignment alone does not tell the student which class is correct. `AT_WEIGHT` sets β in both attention objectives.

Checkpoint — response versus feature transfer

Which comparison of the table of Section 3.1 isolates response transfer, and which isolates feature transfer? Why can the attention distance decrease without the accuracy improving? Why can an attention method cost more during training and the same as CE-only once deployed?

4 Analysis and controlled studies

```
# container
TAG=your_mnist_kd bash scripts/08_analyze_reference.sh
```

Plots (learning curves, weighted losses, attention maps) in `out/runs/<TAG>/plots/`, then the comparison of all models.

It ends with a **reference check** (`summary/reference_gate.txt`): teacher at least 3 points above CE, CE+KD not below CE, the same graph, initial weights, teacher, split, optimizer and stopping rule for every student, T^2 on, CE+KD+AT closer to the teacher's attention than CE. A FAIL is a result to explain, not an error.

Controlled studies

Optional, or at home. Change one factor at a time, in a new TAG that copies the guided teacher. The guided setting measures the KD objective and the extra teacher-labelled data together; to give both methods the same images, set up the TAG with `TRANSFER_SAMPLES=1000`, as in the final task.

```
# container: example, temperature 4
TAG=your_mnist_T4 T=4 TEACHER_SOURCE=copy SOURCE_TAG=your_mnist_kd \
  bash scripts/00_setup_experiment.sh
TAG=your_mnist_T4 bash scripts/03d_copy_teacher_from_tag.sh
TAG=your_mnist_T4 bash scripts/05_cache_teacher_targets.sh
TAG=your_mnist_T4 bash scripts/04_train_student_ce.sh
TAG=your_mnist_T4 bash scripts/06b_train_student_ce_kd.sh
TAG=your_mnist_T4 bash scripts/08_analyze_reference.sh
```

Checkpoint — isolate what caused the gain

Suppose KD outperforms CE. Which controls do you check before attributing the difference to the KD objective? How does your interpretation change because KD used 10,000 teacher-supervised images and CE only 1,000 labelled ones? How do you tell faster convergence from a better validation-selected final model?

5 Deployment on the KV260

The Vitis AI flow is the one of the previous labs; here it checks whether the training objective changes the deployed accuracy, and how much the student, with 10 times fewer MACs, gains on the board.

```
# container
TAG=your_mnist_kd bash scripts/09_prepare_host_comparison.sh
TAG=your_mnist_kd bash scripts/10_quantize_compile_all.sh
TAG=your_mnist_kd bash scripts/11_copy_all_to_board.sh
```

`09` draws a 1,000-image test bundle (float, PTQ and board accuracies) and profiles every model on the host; `10` quantizes and compiles the six models; `11` copies them with the bundle and the board scripts to `/home/root/c5/TP04/<TAG>` (a rerun replaces that folder).

```
# KV260 (ssh root@${BOARD#*@}: BOARD without a user@ part, as in Lab 2)
cd /home/root/c5/TP04/your_mnist_kd
bash 12_run_all_fpga_variants_on_board.sh .          # accuracy, E2E/VART FPS (1 min)
bash 12b_profile_energy_all_variants_on_board.sh .   # power, energy (3 min)
bash 12c_dpu_profile_on_board.sh .                   # xdputil, vaitrace (5 min)
```

```
# container
TAG=your_mnist_kd bash scripts/12_fetch_and_summarize.sh
```

The summary gives the float, PTQ and board accuracy on the same 1,000 images (one image is 0.1 point), the throughputs and the energy. The students deploy the same graph: same cost within noise, different accuracy.

Where does the time go on the KV260?

The teacher has 10 times more MACs than the student. Is it 10 times slower on the board? The four FPS numbers of L02 and the course tools guide: **E2E FPS** (the Python application) and **VART FPS** (`execute_async` and `wait`) from `12`; `xdputil benchmark` with 1 and 4 threads and the DPU time of `vaitrace` (`HW_RT`) from `12c` (teacher and CE+KD student only, since the students share one graph); the baseline-subtracted SOM energy of the repeated-input workload from `12b`. The sensor excludes the carrier and USB camera; this is not isolated DPU energy. Keep its sample counts, units and workload interval with the result. `12_fetch_and_summarize.sh` adds a *time per input* table.

Model	MACs	E2E ms	VART ms	`xdputil` 1T ms	`xdputil` 4T FPS	DPU ms	mJ/input
Teacher							
Student CE+KD							

Checkpoint — where does the time go?

1. Compare the teacher/student ratio of MACs with the ratios of the DPU time, of the `xdputil` FPS and of the E2E FPS. Which ratio is closest to the MAC ratio, and which is close to 1?
2. For one input, how much of the E2E time is DPU time? List what takes the rest.
3. Why is `xdputil` with 4 threads faster than with 1 thread, on the same DPU?
4. For networks of this size, what would raise the application FPS more: a smaller network, or a leaner software path (C++, several threads, batching, pre-processing in the programmable logic)? Roughly when would the DPU time start to dominate?
5. The energy per input falls much more than the E2E time. Why?

Checkpoint — training objective versus deployed cost

Why are the teacher, the KL divergence and the attention losses absent from the XModel? Should a KD student be faster than the CE student? Which quantities of the final table are consequences of the training method, and which of the deployed architecture?

6 Your code: the KD primitives, then the final task

6.1 C5 — implement the primitives

Complete the TODOs of `guided_kd/student/kd_primitives.py`: softened probabilities, response KD, normalized attention map, and the weighted objective. Then check them, and rerun the whole numerical suite when needed:

```
# container
TAG=your_mnist_kd bash scripts/02f_check_primitives.sh
TAG=your_mnist_kd bash scripts/02e_run_concept_suite.sh     # C1 to C4 again
```

The checks test behaviour, not coding style: normalization, softening, teacher detachment, attention-map shape and normalization, and the decomposition of the objective. They do not say whether a hyperparameter setting will improve accuracy.

Checkpoint C5 — connect mathematics to PyTorch

For each line of your implementation, give the mathematical operation. In particular: the class dimension of the softmax, the argument order of the KL divergence, the purpose of `batchmean`, where the teacher is detached, and why only the KD term receives T^2.

6.2 Final task — implement and test response KD

The common runner provides data loading, model construction, validation, early stopping, checkpoint selection, the single test evaluation and the result tables; the KD loss and the training step are yours. The checks test observable behaviour, not one particular source code.

Unguided task — implement and test response KD

Work in `challenge_kd/student/kd_experiment.py`. Do not import `src/kd_losses.py` and do not copy `src/train_student.py`. Implement:

1. `response_kd_loss(student_logits, teacher_logits, temperature)`: softened response KD with teacher detachment and the T^2 factor; it must raise an error for $T \leq 0$;
2. `training_step(...)`: student forward pass, CE on the examples that have a label (given by `label_mask`) and KD on all of them, backward pass and optimizer step, with the teacher left unchanged. Return the loss, alone or first in a tuple, or in a dictionary with optional logits and diagnostics (the docstring lists the accepted forms). The method name is passed through, so you may support a method of your own besides `ce` and `ce_kd`.

Create a same-data experiment (CE and KD on the same 1,000 labelled images) that reuses your validated teacher, then run the behavioural checks:

```
# container
TAG=your_challenge TRANSFER_SAMPLES=1000 \
TEACHER_SOURCE=copy SOURCE_TAG=your_mnist_kd \
  bash scripts/00_setup_experiment.sh
TAG=your_challenge bash scripts/03d_copy_teacher_from_tag.sh
TAG=your_challenge bash scripts/13_check_challenge.sh
```

The checks compute $T^2 D_{KL}(p_t^{(T)} \| p_s^{(T)})$ on a small tensor and test teacher detachment, student gradients, the temperature scale, the student update, the unchanged teacher, CE independent of the teacher, CE+KD sensitive to the teacher, and the limits obtained with a zero CE or KD weight. An implementation that ignores KD, reverses the KL, ignores the weights or updates the teacher fails.

Then train both controls through the common runner and validate the evidence:

```
# container
TAG=your_challenge METHOD=ce bash scripts/13_run_challenge.sh
TAG=your_challenge METHOD=ce_kd bash scripts/13_run_challenge.sh
TAG=your_challenge bash scripts/13_validate_challenge.sh
```

The validation checks that both runs used the same graph, initialization, data split, optimizer, stopping rule and teacher checkpoint. It does not decide whether your conclusion is right.

Required interpretation. Does the distilled student improve the best validation accuracy, the test accuracy, the convergence speed, or none of these? Different selected epochs are not unfair: both runs used the same validation-based stopping rule. A fixed-compute claim needs a separate budget-matched study.

Independent extension. Create another TAG that changes exactly one factor, for example the temperature, the CE/KD balance, the student width, the labelled or transfer budget, the optimizer or the seed. The runner reads every setting from the TAG and refuses overrides on `13_run_challenge.sh`, so the factor is set at setup:

```
# container
TAG=your_challenge_T4 T=4 TRANSFER_SAMPLES=1000 \
TEACHER_SOURCE=copy SOURCE_TAG=your_mnist_kd \
  bash scripts/00_setup_experiment.sh
TAG=your_challenge_T4 bash scripts/03d_copy_teacher_from_tag.sh
```

Reuse the same teacher unless the teacher is your factor. Write a falsifiable prediction before running it, run both CE and CE+KD, and use the complete result table of the new TAG.

7 QCM

For each question, choose the single best answer.

QCM A — soft targets and response distillation

1. A teacher soft target is potentially more informative than a one-hot target because it:
   A. increases the number of classes;
   B. describes relative plausibility among non-target classes;
   C. removes the need for labels in every KD setting;
   D. guarantees that the teacher is correct.
2. When the temperature T is increased while logits are fixed:
   A. the argmax necessarily changes;
   B. the distribution generally becomes softer while the logit ranking is preserved;
   C. all probabilities become exactly equal for any $T > 1$;
   D. the teacher network becomes larger.
3. In the usual PyTorch implementation of response KD, the first argument of `F.kl_div` is commonly:
   A. teacher one-hot labels;
   B. student probabilities without a logarithm;
   C. student log-probabilities at temperature T;
   D. teacher logits without softmax.
4. The main purpose of multiplying the softened KL term by T^2 is to:
   A. make the teacher more accurate;

B. compensate the dominant reduction in gradient scale caused by temperature softening;

C. force KD to dominate CE;

D. normalize the loss by the number of classes.

QCM B — objectives, gradients, and feature transfer

5. Why is ground-truth CE commonly retained in CE+KD?

   A. It anchors learning to the true labels when the teacher is imperfect.

   B. It is required for the teacher to run in evaluation mode.

   C. It changes the deployed student architecture.

   D. It makes temperature unnecessary.

6. Two loss terms have coefficients 0.3 and 0.7. Which statement is correct?

   A. The second term always contributes exactly 70% of the gradient.

   B. Coefficients alone do not determine contribution because raw loss and gradient scales differ.

   C. The first term can never dominate optimization.

   D. The coefficients are irrelevant once Adam is used.

7. Attention transfer is useful when teacher and student channel counts differ because it typically:

   A. copies every teacher channel into the student;

   B. collapses channels into a normalized spatial map;

   C. requires teacher and student to have the same number of parameters;

   D. removes all spatial information.

8. In classical offline KD, the teacher should normally be evaluated under `no_grad` or detached because:

   A. the teacher is the fixed supervision source rather than an optimized model;

   B. gradients through the teacher always reduce memory use;

   C. the student cannot otherwise compute CE;

   D. detachment changes the teacher softmax temperature.

QCM C — fair experiments and deployment interpretation

9. KD has higher accuracy than CE after exactly 10 epochs. The strongest justified conclusion is initially that:

   A. KD always has a better final solution;

   B. the teacher is perfectly calibrated;

   C. KD is better under that particular training budget; convergence speed may be part of the effect;

   D. CE can never catch up with more training.

10. To isolate the effect of the KD objective itself as closely as possible, CE and CE+KD should first be compared with:

    A. different student widths chosen to maximize each method;

    B. the same student graph, initialization, examples, validation protocol, and comparable optimization budget;

    C. a different teacher for each method;

    D. test accuracy used to tune every hyperparameter.

11. If CE, CE+KD and CE+KD+AT all train the same student architecture, which statement about deployment is correct?

A. the KD losses and teacher are automatically embedded in the XModel;

B. the attention maps increase the deployed parameter count;

C. deployment contains the trained student graph; training-only teacher/loss machinery disappears;

D. KD always increases DPU FPS.

12. Two student variants have identical compiled graphs but different trained weights. On the same DPU, one should generally expect:

A. potentially different accuracy but broadly similar graph-level compute cost and throughput;

B. identical accuracy because the graph is identical;

C. the KD variant to execute the teacher on the board;

D. the attention variant to require extra FPGA feature-map outputs.

A Supplied teachers

Teacher checkpoints are supplied for three datasets, so that a more expensive dataset can be studied without training its teacher during the lab. The MNIST one replaces the guided teacher if it cannot be trained: set up the guided TAG with `PROVIDED_TEACHER=mnist_lenet_w1` and run `03b` instead of `03`.

Checkpoint	Teacher	Recommended student	Comment
`mnist_lenet_w1`	LeNet ×1	LeNet student ×1	fallback for the guided path
`fashionmnist_lenet_w1`	LeNet ×1	LeNet student ×1	optional Fashion-MNIST study
`cifar10_tinycnn_w1`	TinyCNN ×1	TinyCNN student ×1	optional CIFAR-10 study; check the teacher's advantage first

```
# container
bash scripts/03a_list_provided_teachers.sh     # catalog, pairings and commands
TAG=your_fashion PROVIDED_TEACHER=fashionmnist_lenet_w1 \
  bash scripts/00_setup_experiment.sh
TAG=your_fashion bash scripts/03b_install_provided_teacher.sh
TAG=your_fashion DOWNLOAD=1 bash scripts/01_prepare_data.sh
```

The setup takes the dataset, the architectures and the recommended student settings from the catalog; the student commands of Section 3 are then the same with this TAG. The CIFAR-10 teacher is a moderate teaching model: interpret KD gains only if it is clearly stronger than the CE student. Training a stronger CIFAR-10 teacher from scratch is possible but is an extension.

References

1. G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," 2015.

2. S. Zagoruyko and N. Komodakis, "Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer," ICLR 2017.

3. PyTorch documentation for softmax, KL divergence, optimizers, checkpointing, and knowledge distillation.

4. AMD Vitis AI documentation matching the course environment for quantization, compilation, XIR, VART, and profiling.